Programming Technology. Shokhrukh Nigmatillaev. Documentation of 1st assignment. Task 1.

Shokhrukh Nigmatillaev	**1. Assignment / Task 1**	10.07.2024
APVAVZ	apvavz@inf.elte.hu	Group 5

Task

There is a race for creatures, which takes place on several consecutive days. **Who wins the race? (In other words, which creature can go farthest and remain live?)** At the beginning, each creature has an **amount of water, and a distance of 0** from the start. There are three different kind days could occur: **sunny, cloudy, rainy**. The movement and the water level of a creature are affected by the type of the day and the creature. At first, a creature changes its water level according to the day, and if it is still alive, it moves. A creature dies if it runs out of water (water level drops to 0 or below). A dead creature doesn't move... Properties of creatures: **name of the creature (string), water level (integer), maximum water level (integer), living (boolean), distance (integer).** The types of creatures on the race are: sandrunner, sponge, walker. The following table contains the properties of the creatures.

	water change			**distance**			**max. water**
	sunny	cloudy	rainy	sunny	cloudy	rainy	
sandrunner	-1	0	3	3	1	0	8
sponge	-4	-1	6	0	1	3	20
walker	-2	-1	3	1	2	1	12

Creatures cannot have water more than their maximum water level. Read the data of the race from a text file. The first line of the file contains the **number of competitors** (lets say N). Each of the following N lines contains a competitor: name, type, initial water level. The properties are separated by spaces; and the type is represented with one character: **r - sandrunner, s - sponge, w - walker**. The last line of the file contains the type of the days on the race: **s - sunny, c - cloudy, r - rainy**. The program should ask for the name of the file, and it has to print out the name of the winner (we can assume that the file is existing and its format is valid).

A possible file content:

```
4
wanderer r 4
walk w 7
slider s 12
sneaky s 10
sccrrrssc
```

Analysis

We have 3 different independent types of creatures in our task. They are: sandrunner, sponge and walker. All of them have name and waterLevel attributes. Also we have 3 different types of days : Sunny, Cloudy, Rainy. Each day depending on the type of day our creature moves and it's waterlevel changes.

Sunny:

Creature	WaterLavel	Distance
sandrunner	-1	3
walker	-4	0
sponge	-2	1

Cloudy:

Creature	WaterLavel	Distance
sandrunner	0	1
walker	-1	1
sponge	-1	2

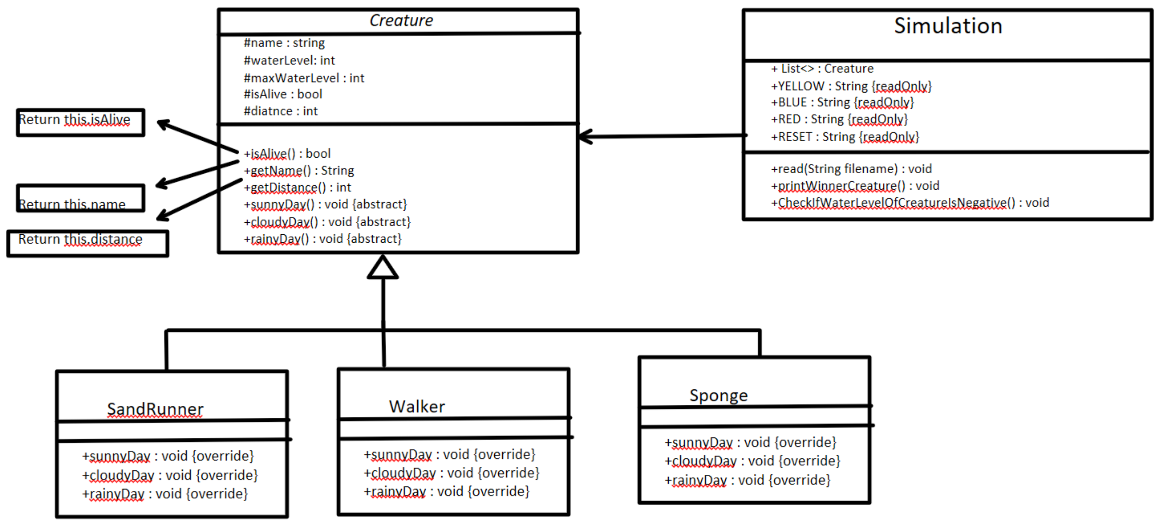
Rainy:

Creature	WaterLavel	Distance
sandrunner	3	0
walker	6	3
sponge	3	1

Plan

In order to describe the whole program with creatures and days I introduced abstract class Creature and 3 children instances inherited from it taking all attributes(String name, int waterLevel, Boolean isAlive and int distance) and methods isAlive(), getDistance(), getName. In the children classes I am overriding 3 abstract methods: void sunnyDay(), void cloudyDay(), void rainyDay().

There is another class – RaceSimulator. Here whole simulation and program runs. We create ArrayList of Creature type. We take filename as an input and then continue reading from a file. While reading from file I introduced some possible error cases that for example amount of creatures cannot be negative number or waterLevel of animals cannot be negative. For waterLevel each creature has its own borders, (0..8], (0..12], (0..20]. So program reads input name and water level of a creature and adds it to list. Then it checks the line with types of days and depending if animal is alive it updates water level and distance. If creature's water level drops down to 0 or lower, it dies and does not continue the way. Then with the help of Maximum search algorithmic pattern we find the creature that traveled the most and print its name.


Creature
#name : string
#waterLevel: int
#maxWaterLevel : int
#isAlive : bool
#diatnce : int
+isAlive() : bool
+getName() : String
+getDistance() : int
+sunnyDay() : void {abstract}
+cloudyDay() : void {abstract}
+rainyDay() : void {abstract}
Return this.isAlive
Return this.name
Return this.distance
Simulation
+ List<> : Creature
+YELLOW : String {readOnly}
+BLUE : String {readOnly}
+RED : String {readOnly}
+RESET : String {readOnly}
+read(String filename) : void
+printWinnerCreature() : void
+CheckIfWaterLevelOfCreatureIsNegative() : void
SandRunner
+sunnyDay : void {override}
+cloudyDay : void {override}
+rainyDay : void {override}
Walker
+sunnyDay : void {override}
+cloudyDay : void {override}
+rainyDay : void {override}
Sponge
+sunnyDay : void {override}
+cloudyDay : void {override}
+rainyDay : void {override}

Short Description of each method

1. Creature **(Abstract class)**

    Members :

        name;
        waterLevel;
        maxWaterLevel;
        isAlive;
        distance;

    Operations :

        isAlive () : to check if creature is alive
        getName() : to receive name
        getDistance() : to receive the distance
        sunnyDay(), cloudyDay(), rainyDay() – abstract methods

2. Sandrunner **(Derived from Creature)**

    Members :

        name;
        waterLevel;
        maxWaterLevel;
        isAlive;
        distance;

    Operations :

        sunnyDay() – if creature is alive it will subtract 1 from water level and if it is still alive it will move for 3
        cloudyDay() – if creature is alive it will move for 1
        rainyDay() – if creature is alive it will add 3 to water level and will not move

3. Sponge **(Derived from Creature)**
   Members :
   name;
   waterLevel;
   maxWaterLevel;
   isAlive;
   distance;
   Operations :
   sunnyDay() – if creature is alive it will subtract 4 from water level and will not move
   cloudyDay() – if creature is alive it will subtract 1 from water level and if it is still alive move for 1
   rainyDay() – if creature is alive it will add 6 to water level and will move for 3
4. Walker **(Derived from Creature)**
   Members :
   name;
   waterLevel;
   maxWaterLevel;
   isAlive;
   distance;
   Operations :
   sunnyDay() – if creature is alive it will subtract 2 from water level and will move for 1
   cloudyDay() – if creature is alive it will subtract 1 from water level and if it is still alive move for 2
   rainyDay() – if creature is alive it will add 3 to water level and will move for 1

RaceSimulator class

Read(String filename) – will read file from input, create objects of corresponding Creatures and add them to Arraylist, then it will go through days and simulate the racing, if file is corrupted or not found, etc it will throw errors.

printWinnerCreature() – will go through all creatures that are alive and with the help of Maximum search check which creature has the most distance covered, then it will print the name of the creature.

CheckIfWaterLevelOfCreatureIsNegative() - it will check if given water level is valid.

Testing

BLACK BOX TESTING:

1. **Valid input test –** provide valid file with the valid amount of creatures, their water levels and types of days (data.txt)
2. **Invalid Input test –** provide invalid input: negative amount of creatures (negativeCreatures.txt), negative water level at the beginning (negativeWaterLevel.txt),

invalid type of creature (invalidCreature.txt), invalid type of day (invalidDay). All these will throw different types of errors.

3. **Empty file –** give empty file to see how program will work (empty.txt)

White Box testing:

1. **InvalidInputException.java –** make sure that all exceptions are handled correctly, all of them are used in the RaceSimulation class.
2. **isAlive()** check if this method returns false when water level becomes < 0
3. **getDistance() –** check if this method returns correct distance of the winner that is still alive.